Activity 3

Activity Overview

During *Activity 3*, you will again hone your coding skills by using arrays to implement another kind of data structure. You will also learn to use runtime exceptions as part of a very standard practice in C# programming: validation of method arguments.

Hints and Tips

Investigate the concept known as a 'circular array' to help you implement queues as arrays.

Activity Instructions

1. First add the existing ***ArrayBase.cs*** (from the finished Activity 2) and ***ArrayQueue.cs*** (from Week 1 Activity Code Files\ folder) to the new empty C# class library project.
2. Examine the class ***ArrayQueue.cs.*** As with ***ArrayStack<T>,*** this class also extends from ***ArrayBase<T>*** and has similar dependencies.
3. There are four methods that need to be completed for ***ArrayQueue<T>*** to work. However, just like ***ArrayStack<T>,*** the parent class must be working properly before you proceed with this activity.
4. A *queue* is another common data structure that follows the FIFO behavior. The first object to enter the queue will be the first object to leave it.

   Note: Refer to the TODO markers in the source file for detailed instructions.
5. The following are methods that you must implement to accomplish the requirements of the activity:
   a. ***public bool Enqueue(T)*** – Places an object at the end of the queue assuming the queue is not full and that the object isn't already in the queue. Returns true if the queue is successful; returns false otherwise. Trying to queue a null object should throw an ***ArgumentNullException***. Remark: To manage queue's *end*, you must use the field **last** and not ***Add()***!
   b. ***public T Dequeue()*** – Removes an object from the start of the queue or throws an ***InvalidOperationException*** if the queue is

empty. Remark: To manage queue's *start*, you must use the field **first** and not ***RemoveAt()***!

c. ***public override int IndexOf(T)*** – Finds the specified object's place in the queue, relative to the *start* of the queue. The start of the queue is place 0. It returns a ***NOT_IN_STRUCTURE*** if the object is not present in the queue. Trying to find a null object should throw an ***ArgumentNullException***.

   Note: The ***override*** concept will be explained later in the course.

d. ***public T Check(int)*** – Determines what object is at the specified index *relative to the start of the queue*. Zero is the start of the queue. Placing an index that is in anyway invalid for the queue it should throw an ***IndexOutOfRangeException***.

6. To test your solution for correctness, add new test project to your existing **Activity3** solution: File > New > Project…. and select Visual C# > Test > Unit Test Project. Name it as **Activity3.Tests** and make sure you are adding it to existing **Activity3** solution.
7. In the **Activity3.Tests** project add test file ***Activity3_Tests.cs*** (from Week 1 Activity Code Files\ folder) and ***PlayerProfile.cs*** (it might be the finished version from Activity 1).
8. In the **Activity3.Tests** project add a reference to the project **Activity3**.
9. In order to activate unit tests open TEST > Windows > Test Explorer and click Run All – if all tests pass green, then your solution is correct. If some tests fail, try to infer why your solution behaves differently.
10. Inform the facilitator upon completion by checking in to the TFS with comment "Finished Activity 3".